In the last lecture we talked about problems that have efficiently verifiable proofs *and* refutations but no efficient algorithms, that is problems in $\mathrm{NP} \cap \mathrm{coNP} \setminus \mathrm{P}$. We saw a few examples of such problems (graph isomorphism, odd number of prime factors, statistical difference) and a general method for turning proofs into (interactive) refutations: the statistical zero-knowledge property.

Refutations are interesting for at least two reasons. First, every algorithm is also a refutation (e.g. Edmonds' algorithm certifies that a graph does not have a perfect matching), so designing a refutation is a prerequisite for designing an algorithm. This perspective is common in approximation algorithms for NP problems, where the first step usually consists of identifying certificate for no instances. For example, it can be shown that if a graph has no vertex cover (a subset of vertices that touches all the edges) of size s then it must have a matching (a set of pairwise disjoint edges) of size more than $s/2$. Thus the existence of a large matching refutes the existence of a small vertex cover.

The other motivation for studying refutations comes from cryptography. For reasons that are not completely understood, security of public-key encryption schemes is usually based on problems that are conjectured to live in $\mathrm{NP} \cap \mathrm{coNP} \setminus \mathrm{P}$. Here is a non-rigorous justification. Suppose we compare the (randomized) encryptions of two messages, let's call them 0 and 1. The security requirement is that an adversary should not be able to distinguish the two, so in particular to decide the promise problem whose YES instances are encryptions of 1 and NO instances are encryptions of 0. Therefore this promise problem (YES, NO) should not be in P. On the other hand, the owner of the secret key *can* tell the two apart efficiently; this is what it means to decrypt. Therefore the secret key is a "certificate" for both the claims "this ciphertext decrypts to 1" and "this ciphertext decrypts to 0", that is it can be used to certify both YES and NO instances, suggesting that (YES, NO) is in $\mathrm{NP} \cap \mathrm{coNP}$.

In summary, identifying new problems in $\mathrm{NP} \cap \mathrm{coNP} \setminus \mathrm{P}$ is of interest to both algorithms and cryptography. Unfortunately there are not so many plausible candidates. One potential avenue of progress is to relax the requirement that refutations are sound in the worst-case (e.g. require the prover to be sound for every possible pair of non-isomorphic graphs) and allow for an incorrect answer with some small probability over the choice of the instance.

1 Errorless and error-prone heuristics

An average-case algorithm for a computational problem is an algorithm that typically succeeds when the instance is chosen at random from some fixed distribution. Two important choices need to be made when defining what it means for an algorithm to typically succeed. The first choice has to do with the fraction of instances on which the algorithm gives the correct answer. In lecture 7 we defined a predictor to be an average-case decision algorithm that has a slightly better-than-random chance of guessing the correct answer. At the other extreme we could require the algorithm to be correct on almost all the inputs.

Unlike the error probability of randomized algorithms, the fraction of inputs on which an efficient algorithm succeeds cannot in general be amplified, so it has to be specified as part of the problem description. In this lecture we will be mainly interested in average-case algorithms that almost always succeed. To be specific let's fix the fraction of instances on which the algorithm is allowed to fail to 1%.

The second choice has to do with the required behavior of the algorithm on those instances that it fails to solve. To illustrate this point let's consider the problem of deciding whether a random graph G on n vertices has a clique of size k. The distribution is the Erdős-Rényi model in which each of the $\binom{n}{2}$ pairs of vertices forms an edge independently with probability half. The expected number of size-k cliques in G is $\binom{n}{k}2^{-\binom{k}{2}}$ (this is the expected sum of the "clique indicators" for each of the $\binom{n}{k}$ potential cliques). Setting $k = \log n$ and using the approximation $\log \binom{n}{k} = k(\log n/k + \Theta(1))$ we get that there are $\Theta(n^{0.5 \log n})$ cliques of size $\log n$ in expectation. A more elaborate second moment calculations shows that G will contain at

least one clique of size $\log n$ with high probability (more than 99%).

An algorithm that accepts every single graph G (outputs yes on every input) will therefore be correct more than 99% of the time. This algorithm is an example of a *heuristic*: In the rare cases when G doesn't have a $(\log n)$-size clique the algorithm will be incorrect. In other words, the algorithm's claim that G doesn't have a clique is not supported by concrete evidence for this specific G. An average-case algorithm that not only decides most inputs correctly but also certifies the correctness of its answer is called an errorless heuristic.

Definition 1. Let (YES, NO) be a decision problem and X be a distribution on instances supported on $YES \cup NO$ of length n. An *errorless heuristic* for (YES, NO) with respect to X is an algorithm A that outputs one of the three values "yes", "no", and "I don't know" such that

1. A is never wrong: event "$x \in YES$ and $A(X)$ says no or $X \in NO$ and $A(X)$ says yes" never happens
2. $A(X)$ says "I don't know" with probability at most 1% over the choice of X.

Heuristics trivially exist in the regime where most of the instances of the problem are YES instances as when deciding whether a random n-vertex graph has a $(\log n)$-size clique. It turns out that there is also an errorless heuristic for deciding if a graph has say a $(0.99 \log n)$-size clique. The algorithm tries to grow a clique by attaching an arbitrary vertex v to a clique recursively constructed on the set of its neighbors. In a random n-vertex graph v will have $(n-1)/2$ neighbors in expectation, so each step roughly halves the size of the problem resulting in a clique of size about $\log n$. This argument can be made rigorous to prove that the result is a clique of size at least $(0.99 \log n)$ with high probability over the choice of graph. If the algorithm finds such a clique it answers yes. Otherwise it answers I don't know. This algorithm is clearly never wrong (it only answers yes when it has seen a clique) and answers I don't know rarely, satisfying both requirements of an errorless heuristic.

When the size of the clique k is between $1.01 \log n$ and $1.99 \log n$, most graphs still have a k-clique, but no efficient errorless heuristics for k-clique are known. When k is larger than $2.01 \log n$ the expected number of k-cliques tends to zero, so most graphs won't have a k-clique. Now a heuristic that answers "no" without looking at the graph will be correct most of the time. Is possible to efficiently decide the coNP property "G has no k-clique" for most G?

It turns out that there is an errorless heuristic for refuting the existence of a k-cliques in a random graph when $k \geq 10\sqrt{n}$. Let A be the adjacency matrix of G with the convention that edges and non-edges are represented by the values 1 and -1, respectively, and the diagonal entries are all ones. Had G contained a size-k clique the spectral norm of A would have been at least k because $v^T A v = k\|v\|^2$ where v is the indicator vector of the clique. It is however known that a random symmetric ± 1 matrix distributed like A has spectral norm at most $10\sqrt{n}$ with high probability. Thus the algorithm that says no if the spectral norm of A is at most $10\sqrt{n}$ and "I don't know" otherwise is an errorless heuristic or k-clique in this regime.

As illustrated by this example, errorless heuristics are in general harder to obtain than error-prone heuristics, and there are specific examples of NP and coNP problems like deciding the existence of a size-k-clique in a random graph for suitable choices of k where a trivial heuristic that doesn't even look at its input works, but no efficient errorless heuristic is known. If P were to equal NP then all NP (and coNP) problems would have worst-case algorithms which are in particular errorless heuristics that never say "I don't know". Thus P $\neq$ NP is a necessary assumption for the existence of NP problems that do not admit efficient errorless heuristics. One of the central open problems in average-case complexity is whether it is also sufficient.

There weren't too many interesting things to say about this question until Shuichi Hirahara made a breakthrough in 2017. To understand the significance of his work it will help to take a detour into a connection between random refutation and Probably Approximately Correct (PAC) learning that was discovered by Daniely, Linial, and Shalev-Schwartz a few years before.

2 Random refutation and hardness of PAC learning

The main task in PAC learning is to predict the value of an unknown function $f(x)$ on a random input x given examples of the form $(x_1, f(x_1)), \ldots, (x_m, f(x_m))$ where $x_1, \ldots, x_m$ are chosen independently at random from some distribution D. For example, suppose after seeing the examples $(5, 10), (19, 38), (17, 34)$ you are asked to predict the ? in $(8, ?)$. Here the samples are consistent with the function $f(x) = 2x$ and most of us would predict 16.

To make sense of PAC learning we must impose some limitation on f. As we discussed in Lecture 8, if f was a random function then it would be impossible to predict even after seeing any number of examples. In PAC learning f is required to be chosen from some hypothesis class $\mathcal{F}$ of "reasonable" functions. The fundamental theorem of PAC learning says that, assuming f is Boolean-valued, $f(x)$ can then be predicted, say with 99% accuracy, given $m = O(\log|\mathcal{F}|)$ random examples. Formally,

Definition 2. Let $\mathcal{F}$ be a class of Boolean-valued functions over the same input domain and D be a distribution on inputs. Algorithm L PAC-learns $\mathcal{F}$ with respect to distribution D from m examples with error at most ϵ if for every $f \in \mathcal{F}$, on inputs $(x_1, f(x_1)), \ldots, (x_m, f(x_m))$ and x, L outputs $f(x)$ with probability $1 - \epsilon$, where $x_1, \ldots, x_m$ and x are chosen independently from D.

Theorem 3 (Fundamental theorem of PAC learning)**.** *For every $\mathcal{F}$ and D and $m = O((\log|\mathcal{F}|/\epsilon^2)$ there exists an algorithm L that PAC learns $\mathcal{F}$ with respect to D from m examples with error at most ϵ.*

Even if all functions in $\mathcal{F}$ have small (some fixed polynomial-size circuits) L may not be efficient: In Lecture 8 we showed that any efficiently computable family $\mathcal{F}$ of pseudorandom functions cannot be learned even if L itself gets to choose the training data $x_1, \ldots, x_m$ and the test x. However we also saw in that lecture that pseudorandom functions cannot be too simple, for example computable by polynomial-size constant-depth circuits because circuit lower bounds for that class are natural.

This leaves open the possibility that if $\mathcal{F}$ is a simple enough class of functions then it can be PAC learned efficiently. One important definitional choice for efficient PAC learning has to do with the distribution D. A distribution-specific learner only needs to learn with respect to a specific distribution D of interest like the uniform distribution. In contrast, a distribution-free PAC learner is required to work for all distributions. One advantage of the distribution-free definition is that it allows *boosting*: If there is an efficient learner L with error $(1 - \epsilon)/2$ (i.e. one that merely predicts a bit better than random) then there is an efficient learner L' with error at most δ with running time polynomial in the running time of L, $1/\epsilon$, and $\log 1/\delta$.

There are few classes of functions that can be distribution-free PAC-learned efficiently. One is the class of perceptrons, that is functions $f\colon \{0,1\}^n \to \{0,1\}$ of the form $f(x) = \text{sign}(w_0 + w_1 x_1 + \cdots + w_n x_n)$ for some real-valued weights $w_0, \ldots, w_n$. Small decision trees can almost be PAC-learned efficiently; the best known learner for size-s decision trees on n inputs takes time $n^{O(\log s)}$. In contrast, DNF formulas cannot be PAC-learned efficiently assuming random k-SAT formulas do not admit efficient errorless heuristics:

Theorem 4. *Assume that for every polynomial t there exists a constant k so that no algorithm running in time $t(n)$ is an errorless heuristic for a random k-SAT formula with n variables and $t(n)$ clauses. Then DNFs cannot be distribution-free PAC-learned in polynomial time.*

The complexity of random kSAT To understand this theorem we need to talk about the average-case complexity of random k-SAT. To be specific let's take $k = 3$. A random 3SAT instance is a 3CNF on n variables in which every clause is chosen independently at random from the following distribution: Choose the three variables uniformly at random among all $\binom{n}{3}$ possibilities. Then independently decide whether to negate each of the variables.

A calculation similar to the one we did for k-clique shows that the expected number of satisfying assignments for a random 3SAT instance is $2^n \cdot (7/8)^m$. When $m > 6n$ this number goes to zero exponentially fast, so most 3SAT instances will be unsatisfiable. For general k most instances become unsatisfiable as soon as m exceeds $2^k n$. This is the relevant regime for Theorem 4.

An errorless heuristic for 3SAT in this regime needs to be certain that the input formula is unsatisfiable before saying so. As for k-clique this problem becomes easier as m becomes larger. For example, as m approaches n^3 we might expect to see all 8 possible clauses with the same literals x_i, x_j, x_k and different negation patterns. An algorithm that looks for such structures would be an errorless heuristic for 3SAT in this regime.

The best known efficient errorless heuristic for random 3SAT kicks in when m is about $n^{1.5}$. When m is between $6n$ and a bit less than $n^{1.5}$ most 3CNFs are unsatisfiable but no efficient errorless heuristics are known. In 2008 Feige, Kim, and Ofek proposed an efficient *nondeterministic* errorless heuristic that works assuming m is larger than about $n^{1.4}$. Their algorithm cannot efficiently certify unsatisfiability on its own, but it can do so for most 3CNFs when provided with a short certificate w. In contrast, no "certificate" can make the algorithm falsely claim that a satisfiable 3CNF is not.

For general k, no polynomial-time errorless heuristic algorithm for certifying unsatisfiability are known when m is between $2^k n$ and about $n^{k/2}$. I don't know what the status is for nondeterministic algorithms but it is seems reasonable to conjecture that efficient errorless heuristics do not exist unless $m > n^{ck}$ for some constant c. This is consistent with the assumption in Theorem 4.

Proof sketch of Theorem 4 As usual we will prove the contrapositive, namely we will show how to use a learner for DNF to refute random kCNFs. To implement this we need to represent a refutation problem as a PAC learning problem Π. It will be helpful to imagine that the input kCNF ϕ is provided in a streaming fashion. There is a "input read" button that when pushed for the first time outputs the first m clauses of ϕ, when pushed the second time outputs the next m clauses of ϕ, and so on. The clauses output at the i-th push of the button are represented by a kCNF ϕ_i so that $\phi = \phi_1 \text{ AND } \phi_2 \text{ AND } \cdots$.

In the initial learning problem Π the kCNFs ϕ_i will play the role of examples and the function f_x to be learned will be indexed by an assignment x. The value $f_x(\phi)$ is the evaluation of ϕ at x, that is true if x satisfies ϕ and false otherwise. To generate examples for Π, in addition to the input ϕ, we sample an auxiliary random k-CNF ρ with its own read button. Each example (ψ_i, y_i) is sampled as follows: With probability $1/2$, ψ_i is generated by pushing the read button for ϕ and setting y_i to one. With the remaining probability, it is generated by pushing the read button for ρ and setting y_i to zero.

If ϕ has a satisfying assignment x and m is sufficiently large ($m = 3k \log t(n)$ suffices), then with high probability all labels y_i will equal $f_x(\psi_i) = \psi_i(x)$: Whenever the read-ϕ button is pushed this is true because x must satisfy the part ψ_i that was output. Whenever read-ρ is pushed, x satisfies ψ_i with probability $(1 - 2^{-k})^m < 1/10t(n)$ because ψ_i is a random m-clause k-CNF. By a union bound, with high probability $\psi_i(x) = 0$ for all ψ_i obtained by a read-ρ push.

If, on the other hand ϕ is a random kCNF, then the label y_i is a random bit that is independent of ψ_i: ϕ and ρ are identically distributed so y_i gives no information about which button was pushed. In summary, we have a reduction with the following properties: If ϕ is satisfiable, then y_i equals $f_x(\psi_i) = \psi_i(x)$ for all i; if ϕ is random, then y_i is independent of ψ_i for all i. A distribution-free PAC learner for Π can distinguish between these two cases.

Lemma 5. *Assume there exists an efficient distribution-free PAC learner L for Π (with error $1/10$). Then there exists an efficient (randomized) distinguisher that accepts all sufficiently long sequences of distinct items of the form $(\psi_i, f_x(\psi_i))$ with probability at least 99% and rejects sequences of the form (ψ_i, b_i) where $b_1, b_2, \ldots$ are independent random bits with probability at least 99%.*

Proof. Suppose the learner requires ℓ examples and the sequence consists of 3ℓ items $(\psi_1, y_1), \ldots, (\psi_{3\ell}, y_{3\ell})$. The learner is provided a random training sample of the items (chosen with repetition) and challenged on a random item ψ_I. If the learner's prediction matches y_i the distinguisher accepts and otherwise it rejects. When $y_i = f_x(\psi_i)$ the learner sees correctly labeled independent examples so it will output the correct prediction $f_x(\psi_I)$ with probability 9/10. On the other hand, when y_i are independent random bits the learner has no information about the value of y_I unless item I was included in the training sample. This happens with probability at most $\ell/3\ell = 1/3$. Overall y_I is predicted correctly with probability at most

$1/3 + 1/2 \cdot 2/3 = 2/3$. By repeating independently over fresh data a few times the success probability becomes 99%. □

Therefore if Π can be learned then it is possible to efficiently distinguish 99% of random kCNFs from 100% of satisfiable ones, which is precisely the job of a (randomized) errorless heuristic. But Π is a strange learning problem in which the examples $\psi_1, \psi_2, \ldots$ are CNFs and the concept to be learned is an assignment. To finish the proof of Theorem 4 we want to reduce Π to the more natural problem of PAC learning a DNF g given training data in the form of assignments x and labels $g(x)$.

This reduction can be implemented in three stages. First, learning CNF is reduced to learning DNF by negating both the examples and labels, i.e (ψ_i, y_i) is mapped to $(\text{NOT } \psi_i, \text{NOT } y_i)$ and the learner's prediction is also negated. Second, learning DNF is reduced to learning *monotone* DNF, namely DNF without negated literals. This is accomplished by replacing every negated variable $\overline{x_i}$ by a new variable x_i^* and extending the hidden assignment x so that every new variable x_i^* is assigned the value NOT x_i.

At this point we have an instance whose examples are monotone DNFs ψ and labels are evaluations $f_x(\psi) = \psi(x)$. Represent the DNF ψ by an $m \times n$ Boolean matrix whose (j,i)-th entry ψ_{ji} indicates the presence of variable x_i in the j-th term of ψ. Checking whether x satisfies ψ amounts to verifying that when the columns indexed by zero-entries of x are dropped this matrix has an all-ones row, namely

$$x \text{ satisfies } \psi \text{ if and only if } \text{OR}_j \text{ AND}_{i:\, x_i=1} \psi_{ji}$$

which is a DNF in the matrix representation of ψ! In summary the evaluation $f_x(\psi)$ is itself a DNF of size at most m in mn inputs, completing the chain of reductions.

3 Average-case complexity

NP-completeness is arguably the most useful "export" of complexity theory to the rest of computer science and beyond. A key reason for its importance is the ubiquity of NP-complete problems from a variety of different domains. This wikipedia page lists about a hundred such problems. If you are unsuccessful in designing an efficient algorithm for some NP problem you are interested in, chances are you will find that problem or a closely related one in the list.

There is no a priori reason why so many of the problems within NP either admit efficient algorithms (they are in P) or are NP-complete. Ladner's Theorem says that non-NP complete problems outside P must exist (unless P equals NP). In practice, however, such problems rarely if ever come up. In this class we had to work fairly hard to construct plausible examples of non-NP-hard problems within NP like graph isomorphism and statistical difference.

Owing to the prevalence of NP-hard problems among those that are not efficiently tractable it is useful to have strategies for coping with NP-hardness. One such strategy is average-case algorithm design: If we have some information about the distribution on inputs we may be able to solve the problem on most instances of interest. This is a natural assumption in statistical inference, machine learning, and cryptanalysis among other areas.

We saw that a worst-case hard problem like CLIQUE may become easy on average under certain distributions, for example when the graph is random and the clique size k is either less than $0.99 \log n$ or greater than $10\sqrt{n}$. Yet it appears to remain computationally intractable under other distributions. Wouldn't it be nice if there was an explanation along the lines of NP-hardness for our inability to find efficient heuristics for problems like k-clique?

In the 1980s Levin identified a class of "average-case NP-complete problems" with the property that if any of these problems admits an errorless heuristic, so does every problem in "distributional NP". (A specification of an average-case problem must also include the distribution on instances. Distributional NP consists of all NP problems with respect to all efficiently samplable distributions.) The list of known distributional NP-complete problems is, however, very short and does not contain most problems that

researchers care about like random k-CLIQUE and random SAT. At this point there is little evidence that these problems are complete for distributional NP.

There is an appealing alternative approach for arguing that a problem is average-case hard: reducing from a worst-case hard problem. We can illustrate this idea on the discrete logarithm problem. A cyclic subgroup of a finite multiplicative group (like the integers modulo n) is a sequence of powers of some generator g. The order of the subgroup is the smallest number n such that g^n equals 1. The discrete logarithm problem with respect to g is, on input h in the subgroup, to find the unique power $0 \leq y < n$ such that $g^y = h$. It turns out that if we can solve this problem for say 1% of the inputs h, then we can solve it for most h. To do so, given an arbitrary input h, we ask to solve the problem on input $h \cdot g^r$ for a random $0 \leq r < n$. The element $h \cdot g^r$ is random in the subgroup so a solution will be produced with 1% probability *over the choice of* r. If this is the case subracting r from this solution will output the discrete log of h. By repeating 500 times independently and checking which answer works the success probability will be amplified to 99%.

This is an example of a *worst-case to average-case reduction*: An algorithm that solves 1% of discrete log instances can be used to solve all instances. Conversely, if we believe that discrete log is worst-case hard, then it is also hard-on-average 99% of the time. If we could similarly reduce from some (worst-case) NP-complete problem A to a distributional problem B we would be getting the best possible evidence that B is average-case hard: If there was a heuristic for B it could be used to solve all instances of A implying that P equals NP.

All known worst-case to average-case reductions, however, start with a worst-case problem in NP$\cap$coNP. There is some evidence that this is no accident: It can be proved that some restricted types of reductions to distributional NP problems must start from a worst-case problem in AM$\cap$coAM. One interpretation of this evidence is that we should not expect to prove distributional NP-hardness from a worst-case hardness assumption in NP $\setminus$ coNP. Hirahara's theorem upends this expectation.

4 Meta-complexity

The *Kolmogorov complexity* $K(x)$ of a string $x \in \{0,1\}^n$ is the size of (length of the string that encodes) the smallest Turing Machine M that halts and outputs x. For example, the string $0101\cdots01$ (repeated n times) has Kolmogorov complexity $\log n + O(1)$: It can be output by a Turing Machine that first writes the number n in binary representation on a helper tape (this takes $\log n + O(1)$ states) then decrements this counter down to zero writing out a 01 in each step (this takes another $O(1)$ states).

This definition is sensitive to the convention used to encode Turing Machines as strings. It happens that there is a "best" encoding in the sense that no encoding can be shorter than it by more than some fixed constant. The reason is that a program written in any language can be simulated on a Turing machine with a fixed-size code snippet that specifies how to carry out the simulation. Therefore the length of a program in say python that outputs x is an upper bound of its Kolmogorov complexity up to a constant additive term. For example the python program "`for i in range(`n`): print(01)`" of size $\lceil \log n \rceil$ (the description of n) plus some constant (the description of the other symbols) certifies that 01 repeated n times has complexity $\log n + O(1)$.[1]

Meta-complexity is the study of the computational complexity of computing complexity measures like K. It turns out that K cannot be computed at all. The following promise problem APXK is undecidable:

Input: A string x
Yes instances: $K(x) \leq \log|x| + \log\log|x|$
No instances: $K(x) > |x| - 2$.

Lemma 6. *The probability that $K(X) \leq n - c$ for a random n-bit string X is at most 2^{-c+1}.*

[1] This is not exactly true because the program characters are unicode symbols while python requires n to be provided in decimal or hexadecimal notation, resulting in a Boolean program representation of length $\Theta(\log n)$.

Proof. There are at most 2^{n-c+1} Turing Machines of size at most $n-c$, so there are at most that many strings that can be output by them. The probability that a random n-bit string is the output of any one of them is at most 2^{-c+1}. □

In particular this lemma says that there exists a string of complexity $n-2$, so APXK has no instances of all lengths. Assume APXK were decidable and consider the lexicographically smallest string x of length n on which the program for APXK says no. This x cannot be a yes instance so it must have complexity more than $\log n + \log\log n$. Oh the other hand, the program "exhaustively search for the smallest length-n string on which the program for APXK says no" outputs x and has description length $\log n + O(1)$, a contradiction!

The reason Kolmogorov complexity is impossible to compute is that it takes into account only the size of the program but not its running time. The *time-bounded Kolmogorov complexity* $K^t(x)$ is the size of the smallest Turing Machine that outputs x in at most t steps (and is undefined if no such Turing Machine exists). Approximating K^t is captured by the promise problem $\text{APXKT}_{\Delta s}$:

Input: A string x and numbers t, s (represented in unary)
Yes instances: $K^t(x) \leq s$
No instances: $K(x) > s + \Delta s$.

Unlike APXK, APXKT is not only decidable but it is an NP problem: The predicate "M has size at most s and outputs x in t steps" can be verified in time linear in the description size of M and t.

Could APXKT also be in P? One reason this is unlikely is that breaking a pseudorandom generator is a special case of APXKT. Suppose $G\colon \{0,1\}^k \to \{0,1\}^n$ is a candidate pseudorandom generator that is time-$(t(k)+k)$ computable by a Turing Machine. Then outputs of G have $t(k)$-time bounded Kolmogorov complexity $k + O(1)$ as each of them can be computed by a program of the form "run G on seed r", so they produce instances of APXKT. On the other hand, a random x is a no instance by Lemma 6, even if Δs is as large as $n - k - O(1)$.

Could APXKT be in coNP? In the proof of Theorem 4 we argued that if random 3SAT has an errorless heuristic if it is possible to distinguish sequences of the form $(\phi_1, \phi_1(x)), \ldots, (\phi_\ell, \phi_\ell(x))$ (type SAT) from sequences of the form $(\rho_1, b_1), \ldots, (\rho_\ell, b_\ell)$, where ρ_i are random 3CNF and b_i are independent bits (type RAND). By Lemma 6, the Kolmogorov complexity of most type-RAND sequences is at least $(d+1)\ell - 10$, where d is the number of random bits it takes to describe each of the formulas ρ_i (for an m-clause, n-variable random 3CNF d is about $3m\log(2n)$). On the other hand, a type SAT sequence can be efficiently generated by a Turing Machine of size $d\ell + n + O(1)$: $\phi_1, \ldots, \phi_\ell$ take $d\ell + O(1)$ bits to describe and x takes an additional n bits. Given this information the sequence can be generated in linear time by a constant-size program. The complexity gap Δs between the two is at least $\ell - n - O(1)$.

If $\text{APXKT}_{\ell-n-O(1)}$ were in coNP, this reduction would produce a nondeterministic errorless heuristic for random 3CNF with n variables and about $\ell\log\ell$ clauses assuming say $\ell > 2n$. Current algorithmic evidence suggests that no such algorithm should exist when $\ell \leq n^{1.39}$.

5 Hirahara's theorem

Hirahara's theorem shows that if APXKT has errorless heuristics then it can be solved in the worst case:

Theorem 7. *If* $\text{APXKT}_{\Delta s}$ *has a polynomial-time errorless heuristic then it has a randomized worst-case polynomial-time algorithm (with* $\Delta s = O((\sqrt{s} + \log|x|)\log|x|)$*).*

Assuming say $s < 0.99|x|$, most instances are no instances of APXT by Lemma 6. An errorless heuristic D for APXKT will reject most of these instances but output "I don't know" on all the ones that have small time-bounded Kolmogorov complexity. Such a heuristic therefore distinguishes low-complexity strings from random ones.

A natural proof strategy for Theorem 7 is to look for a reduction that maps low-complexity strings into low-complexity strings and high-complexity strings into random strings. It is unclear how complexity can be turned into randomness, and the negative results on worst-case to average-case reduction we mentioned suggest this may be challenging. However we already saw a transformation that almost does the job: The Nisan-Widgerson generator turns computational hardness (another form of complexity) into *pseudo*randomness.

Theorem 7 is indeed proved by interpreting the worst-case MINKT instance f as the truth-table of a function and outputting the result y of the Nisan-Wigderson generator applied to f on a random seed. This transformation outputs $y = (f(x|_{S_1}), \ldots, f(x|_{S_m}))$ for a random string x. Recall that $S_1, \ldots, S_m$ is a combinatorial design, that is a large collection of large sets every pair of which have relatively small intersection. If f has low time-bounded Kolmogorov complexity and the design can be efficiently computed in a suitable sense then so will g.

On the other hand, the proof of Nisan and Wigderson showed that if g can be distinguished from a random function by the errorless heuristic D then f can be predicted on random inputs using D and some extra information (the truth-tables of all $\binom{m}{2}$ restrictions of f on subsets of size $t_\cap$). Thus f can be approximated by some function of low Kolmogorov complexity. By using a more clever encoding of the truth-table of f it can be ensured that f has low Kolmogorov complexity itself. Apart from bookkeeping complexity losses and the choice of a suitable design this is the whole proof of Hirahara's theorem.

An intriguing follow-up question is whether APXKT is an NP-complete problem. Proving this would establish the existence of distributional NP problems that are NP-hard to solve by errorless heuristics. Hirahara himself proved that some related problems are NP-complete. A few weeks ago Ilango gave strong evidence that a related problem called Minimum Circuit Size which also admits worst-case and average-case equivalence is likely to be NP-complete, but his argument for NP completeness relies on an unproven average-case hardness assumption. The plot is getting thicker.

References

The study of average-case complexity for NP was initiated in a legendary 2-page paper of Levin. The difficulty of applying Levin's theory to natural problems was addressed by Trevisan. The distinction between error-prone and errorless heuristics was pointed out in another legendary survey by Impagliazzo. Impagliazzo's paper laid out the main complexity theoretic questions which largely remain open.

The complexity of random kSAT and other constraint satisfaction problems was studied by several authors. This survey of Allen, O'Donnell, and Witmer is a good starting point. A good source for PAC learning is the book of Shalev-Schwartz and Ben-David.

Theorem 4 was proved by Daniely, Linial, and Shalev-Schwartz. The exposition here is based on this paper of Vadhan.